View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Elsevier - Publisher Connector

North-Holland

EDITORIAL

Past research on mathematical foundations of computer science has focused mostly on the study of the mathematics of software objects and little has been done to develop software objects on a mathematical basis. Therefore, there are voices doubting the usefulness of a mathematical foundation for software development based on such things as universal algebra and category theory. But no other viable alternative has been discovered to help in creating a software technology based on mathematics rather than on ad hoc developments. Moreover, algebraists and computer scientists have begun to relate the abstractions in computer science to the process of abstract representation in universal algebra. A strong trend of applying universal algebras as the mathematical foundation of computer science is in vogue. Languages, programs, and processes have been identified as fundamental objects in the study of computer science as a discipline, and abstract models characterizing them have been developed.

In pursuing this trend we need to observe however that there are differences between the objects and methods used in universal algebra and computer science. While abstractions used in universal algebra represent the behavior of ideal (mathematical) objects, abstractions in computer science represent the behavior of real (physical) objects. While the ideal character of the abstractions in universal algebra allows systematic approaches of their specifications and the development of formal notations naturally suited to handling them, computer science still develops formal notations to denote real objects that are rarely formally specified. While an algebraic language accommodates *semantics*, *syntax*, and the *semantics* $\leftrightarrow$ *syntax* association of the algebraic abstractions naturally, the *syntax* and the *semantics* of a computer language are still specified by different mechanisms and their association in a language is artificial.

The similarities between the abstractions handled in universal algebra and computer science lead to the development of new mathematical theories. Our conjecture is that keeping in view also the differences between the abstractions used in universal algebra and computer science, new mathematics can be created that will facilitate the construction of the (software) objects arising in computer science. The goal of the first international *Conference on Algebraic Methodology and Software Technology*, *AMAST*, held on May 22–24, 1989 in Iowa City, IA, was to consolidate this conjecture, looking at algebraic methodology as a foundation for software technology and showing that universal algebra provides a practical mathematical alternative to the ad hoc approaches used in software development. Therefore, unlike other conferences on mathematical foundations of computer science, in which the mathematics is usually enriched with new theories originated in computer science,

0304-3975/90/$03.50 © 1990—Elsevier Science Publishers B.V. (North-Holland)

the submissions to this conference indeed show developments in computer science that originate in mathematics.

Of the eighty-nine papers submitted in response to the call for papers, twenty-seven were selected for presentation at the conference and were published in the proceedings. Of the twenty papers submitted to the issue of *Theoretical Computer Science* dedicated to this conference, we selected the seven most representative for publication in this issue and encouraged another eleven to be resubmitted to *Theoretical Computer Science* as individual contributions.

In this issue, L. Bradley in "Abstract Language Design" identifies the problem of constructing programs from meanings as the fundamental problem of a language user. However, the conventional language design teaches the user how to associate meanings with programs. Hence, Bradley develops an algebraic framework in which a programming language grows iteratively from semantic requirements rather than from syntactic restrictions. H. Ehrig, W. Fey, H. Hansen, M. Löwe, D. Jacobs and F. Parisi-Presicce, in "Compatibility Problems in the Development of Algebraic Module Specification", approach the mathematical model of software development. Their model provides an algebraic framework in which software can be layered in a hierarchy. Each layer of this hierachy is constructed as an algebraic system in terms of modules as components and a given number of module operations (composition, actualization, union). Refinement simulation, and realization are the major operations that allow the layers to be constructed on top of each other. S. Even and D. Schmidt in "Category-Sorted Algebra-Based Action Semantics" define a model for action semantics based on category-sorted algebras and develop a unification-based decidable type inference algorithm for action semantics. R. Janicki and T. Muldner, in "Sequential Specifications Equivalent to Concurrent Specifications" provide a mathematical model for parallel programming which allows the mapping of sequential programs into equivalent parallel programs. A specific of this model is that it allows a user-controlled parallel program development. V. Manca, A. Salibra and G. Scollo in "Equational Type Logic" develop a type algebra which provides a unified approach for treating diverse computation phenomena such as partiality, type polymorphism and dependent types. D. Pigozzi in "Data Types over Multi-Valued Logics" develops an algebraic framework for the logical languages used to reason about abstractions in computer science. The formal mapping of such a language into a universal algebra provides a rationale for using universal algebra as the foundation in computer science. E.G. Wagner in "An Algebraically Specified Language for Data Directed Design" approaches a mathematical treatment of programming languages in which the semantics of the language is developed before committing to any particular syntax or language implementation. An illustration of this approach is provided by designing a language for object oriented programming.

While all these papers clearly show how one can use algebraic methodology for the development of software technology, they also show that we are still far from fully attaining this goal. The major results of these papers are still theorems about computing abstractions rather than theorems about the functionality of these abstractions in a given computer environment. However, by reading these papers, one may

get the feeling that the papers that will be presented at the next *AMAST Conference* will bring us much closer to the goal of transforming algebraic methodology into software technology.

I would like to acknowledge the efforts of all the authors and referees and to thank M. Nivat, Editor-in-Chief of *Theoretical Computer Science*, for helping us bring AMAST's idea to light.

TEODOR RUS
Programme Chairman and
Guest-Editor